

```

#include <stdio.h>
#include <stdlib.h>

typedef struct Node
{
    int data;
    struct Node *lptr;
    struct Node *rptr;
} Node;

Node* createNode(int value)
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    if(newNode==NULL)
    {
        printf("The memory allocation is not successfull\n");
        exit(0);
    }
    newNode->data = value;
    newNode->lptr = NULL;
    newNode->rptr = NULL;
    //printf("The new node has been created successfull\n");
    return newNode;
}

int getCount(Node* head)
{
    int count = 0;
    while (head != NULL)
    {
        count++;
        head = head->rptr;
    }
    return count;
}

Node* insertFront(Node* head, int value)
{
    Node* newNode = createNode(value);
    if (head != NULL)
    {
        newNode->rptr = head;
        head->lptr = newNode;
    }
    return newNode;
}

Node* insertEnd(Node* head, int value)
{
    Node* newNode = createNode(value);
    if (head == NULL)
        return newNode;

    Node* temp = head;
    while (temp->rptr != NULL)
        temp = temp->rptr;

    temp->rptr = newNode;
    newNode->lptr = temp;
    return head;
}

Node* insertAtPos(Node* head, int value, int pos)
{
    int count = getCount(head);

```

```

if (pos < 1 || pos > count + 1)
{
    printf("Invalid position!\n");
    return head;
}

Node* newNode = createNode(value);
if (pos == 1)
{
    if (head != NULL)
    {
        newNode->rptra = head;
        head->lptra = newNode;
    }
    return newNode;
}

Node* temp = head;
for (int i = 1; i < pos - 1; i++)
    temp = temp->rptra;

newNode->rptra = temp->rptra;
newNode->lptra = temp;

if (temp->rptra != NULL)
    temp->rptra->lptra = newNode;

temp->rptra = newNode;

return head;
}

Node* insertByOrder(Node* head, int data)
{
    Node* newNode = createNode(data);

    if (head == NULL || data <= head->data)
    {
        newNode->rptra = head;
        if (head != NULL)
        {
            head->lptra = newNode;
        }
        return newNode;
    }

    struct Node* current = head;
    while (current->rptra != NULL && current->rptra->data < data)
        current = current->rptra;

    newNode->rptra = current->rptra;
    newNode->lptra = current;

    if (current->rptra != NULL)
        current->rptra->lptra = newNode;

    current->rptra = newNode;
    return head;
}

Node* deleteFront(Node* head)
{
    if (head == NULL)

```

```

    {
        printf("List empty!\n");
        return NULL;
    }
    Node* temp = head;

    if (head->rptra != NULL)
    {
        head = head->rptra;
        head->lptra = NULL;
    }
    else
    {
        head = NULL;
    }
    printf("The node %d has been deleted from the beginning of the linked list
successfully\n", temp->data);
    free(temp);
    return head;
}

Node* deleteEnd(Node* head)
{
    if (head == NULL)
    {
        printf("List empty!\n");
        return NULL;
    }
    Node* temp = head;
    if (temp->rptra == NULL)
    {
        free(temp);
        return NULL;
    }

    while (temp->rptra != NULL)
        temp = temp->rptra;

    temp->lptra->rptra = NULL;
    printf("The node %d has been deleted from the end of the linked list
successfully\n", temp->data);
    free(temp);
    return head;
}

Node* deleteAtPos(Node* head, int pos)
{
    int count = getCount(head);
    if (head == NULL)
    {
        printf("List empty!\n");
        return NULL;
    }
    if (pos < 1 || pos > count)
    {
        printf("Invalid position!\n");
        return head;
    }

    Node* temp = head;

    if (pos == 1)
    {
        if (head->rptra != NULL)

```

```

        {
            head = head->rptr;
            head->lptr = NULL;
        }
        else
        {
            head = NULL;
        }
        printf("The node %d has been deleted from the position %d of the linked
list successfully\n", temp->data, pos);
        free(temp);
        return head;
    }
    for (int i = 1; i < pos; i++)
        temp = temp->rptr;

    temp->lptr->rptr = temp->rptr;

    if (temp->rptr != NULL)
        temp->rptr->lptr = temp->lptr;

    printf("The node %d has been deleted from the position %d of the linked
list successfully\n", temp->data, pos);
    free(temp);
    return head;
}

Node* deleteByKey(Node* head, int key)
{
    if (head == NULL)
        return NULL;

    Node* current = head;

    while (current != NULL && current->data != key)
        current = current->rptr;

    if (current == NULL)
    {
        printf("Node with key %d not found.\n", key);
        return head;
    }

    if (current == head)
    {
        head = current->rptr;
        if (head != NULL)
            head->lptr = NULL;
        free(current);
        return head;
    }
    if (current->lptr != NULL)
        current->lptr->rptr = current->rptr;

    if (current->rptr != NULL)
        current->rptr->lptr = current->lptr;

    printf("The node %d has been deleted from the linked list
successfully\n", current->data);
    free(current);
    return head;
}

Node* reverseList(Node *head)

```

```

{
    if (head == NULL)
        return NULL;

    struct Node *temp = NULL, *current = head;

    while (current != NULL)
    {
        temp = current->lptr;
        current->lptr = current->rptra;
        current->rptra = temp;
        current = current->lptra;
    }
    // After the loop, temp is at the previous node of NULL So the new head is
temp->lptra
    if (temp != NULL)
        head = temp->lptra;

    printf("The reverase operation on Double Linked List has been
successfull\n");
    return head;
}

Node* copyList(Node *head)
{
    if (head == NULL)
        return NULL;

    Node *newHead = NULL, *tail = NULL, *temp = head;
    while (temp != NULL)
    {
        Node *newNode = createNode(temp->data);
        if (newHead == NULL)
        {
            newHead = newNode;
            tail = newNode;
        }
        else
        {
            tail->rptra = newNode;
            newNode->lptra = tail;
            tail = newNode;
        }
        temp = temp->rptra;
    }
    printf("The copy operation on Double Linked List has been successfull\n");
    printf("The address of the old Double Linked List is %p\nThe address of the
new Double Linked List is %p\n", head, newHead);
    return newHead;
}

Node* searchByKey(Node* head, int key)
{
    if (head == NULL)
    {
        printf("List is empty. Cannot search.\n");
        return NULL;
    }
    Node* current = head;
    while (current != NULL)
    {
        if (current->data == key)
        {
            printf("Key %d found\n", key);

```

```

        return current;
    }
    current = current->rptra;
}
printf("Key %d not found in the list.\n", key);
return NULL;
}

void display(Node* head)
{
    if (head == NULL)
    {
        printf("List empty!\n");
        return;
    }

    Node* temp = head;
    while (temp != NULL)
    {
        printf("%d <-> ", temp->data);
        temp = temp->rptra;
    }
    printf("NULL\n");
}

int main()
{
    Node* head = NULL;
    int choice, value, pos, key;

    while (1)
    {
        printf("\n--- DOUBLY LINKED LIST MENU ---\n");
        printf("1. Insert Front\n");
        printf("2. Insert End\n");
        printf("3. Insert At Position\n");
        printf("4. Insert By Order\n");
        printf("5. Delete Front\n");
        printf("6. Delete End\n");
        printf("7. Delete At Position\n");
        printf("8. Delete By Key\n");
        printf("9. Reverse Double Linked List\n");
        printf("10. Copy Double Linked List \n");
        printf("11. Search By Key \n");
        printf("12. Display\n");
        printf("13. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                head = insertFront(head, value);
                printf("The node %d has been inserted at the beginning\n", value);
                break;

            case 2:
                printf("Enter value: ");
                scanf("%d", &value);
                head = insertEnd(head, value);
                printf("The node %d has been inserted at the end\n", value);
                break;

            case 3:
                printf("Enter position: ");
                scanf("%d", &pos);
                if (pos < 1 || pos > 100)
                {
                    printf("Invalid position\n");
                    continue;
                }
                printf("Enter value: ");
                scanf("%d", &value);
                head = insertAtPosition(head, value, pos);
                printf("The node %d has been inserted at position %d\n", value, pos);
                break;

            case 4:
                printf("Enter value: ");
                scanf("%d", &value);
                head = insertByOrder(head, value);
                printf("The node %d has been inserted in the list\n", value);
                break;

            case 5:
                head = deleteFront(head);
                printf("Node deleted from front\n");
                break;

            case 6:
                head = deleteEnd(head);
                printf("Node deleted from end\n");
                break;

            case 7:
                printf("Enter position: ");
                scanf("%d", &pos);
                if (pos < 1 || pos > 100)
                {
                    printf("Invalid position\n");
                    continue;
                }
                head = deleteAtPosition(head, pos);
                printf("Node deleted from position %d\n", pos);
                break;

            case 8:
                printf("Enter key: ");
                scanf("%d", &key);
                head = deleteByKey(head, key);
                printf("Node deleted by key %d\n", key);
                break;

            case 9:
                head = reverseList(head);
                printf("List reversed\n");
                break;

            case 10:
                head = copyList(head);
                printf("List copied\n");
                break;

            case 11:
                printf("Enter key: ");
                scanf("%d", &key);
                if (search(head, key))
                {
                    printf("Key %d found in the list\n", key);
                }
                else
                {
                    printf("Key %d not found in the list\n", key);
                }
                break;

            case 12:
                display(head);
                break;

            case 13:
                return 0;
        }
    }
}

```

```

        break;
    case 3:
        printf("Enter value: ");
        scanf("%d", &value);
        printf("Enter position: ");
        scanf("%d", &pos);
        head = insertAtPos(head, value, pos);
        printf("The node %d has been inserted at the specified
position successfully\n",value);
        break;
    case 4:
        printf("Enter value: ");
        scanf("%d", &value);
        head=insertByOrder(head,value);
        printf("The node %d has been inserted in the order
successfully\n",value);
        break;
    case 5:
        head = deleteFront(head);
        break;
    case 6:
        head = deleteEnd(head);
        break;
    case 7:
        printf("Enter position: ");
        scanf("%d", &pos);
        head = deleteAtPos(head, pos);
        break;
    case 8:
        printf("Enter the key: ");
        scanf("%d", &key);
        head = deleteByKey(head, key);
        break;
    case 9:
        head=reverseList(head);
        break;
    case 10:
        head=copyList(head);
        break;
    case 11:
        printf("Enter the key: ");
        scanf("%d", &key);
        searchByKey(head, key);
        break;
    case 12:
        display(head);
        break;
    case 13:
        printf("Exiting.....\n");
        exit(0);
    default:
        printf("Invalid choice!\n");
    }
}
return 0;
}

```